Paleti Krishnasai
CED18I039

Operating Systems - FORK_EXEC assignment-3

1.)Test Drive all the examples discussed so far in the class for the usage of wait, exec call variants.

execl()

In execl() system function takes the path of the executable binary file (i.e. **/bin/ls**) as the first and second argument. Then, the arguments (i.e. **-lh**, **/home** or current directory) that you want to pass to the executable followed by **NULL**. Then execl() system function runs the command and prints the output. If any error occurs, then execl() returns -1. Otherwise, it returns nothing.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int main()



{
   pid_t pid; //this is to use the pid data type
   pid = fork();
   if(pid == 0)
       execl("/bin/ls", "/bin/ls", "-lh", NULL); //child image is now ls
command
   else
   {
       wait(NULL); // parent waits for the child to complete execution
       printf("Parent process gets control \n");
       printf("Parent has waited for child to complete\n");
   }
}
```

Paleti Krishnasai
CED18I039

Output :

```
paleti@paletil:~/OS_LAB$ cd fork_2
paleti@paletil:~/OS_LAB/fork_2$ make execl
cc     execl.c   -o execl
paleti@paletil:~/OS_LAB/fork_2$ ./execl
total 316K
-rwxrwxr-x 1 paleti paleti  13K Sep  8 21:47  a.out
-rw-rw-r-- 1 paleti paleti 1.3K Sep  8 01:32  armstrong.c
-rwxrwxr-x 1 paleti paleti 8.5K Sep 17 00:52  armstronggen
-rw-rw-r-- 1 paleti paleti 1.1K Sep 17 00:52  armstrong_gen.c
-rw-rw-r-- 1 paleti paleti 2.0K Sep  8 21:47  ascdescsort.c
-rw-rw-r-- 1 paleti paleti  197 Sep 18 00:44  bsearch.cpp
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 00:46  execl
-rw-rw-r-- 1 paleti paleti  487 Sep 18 00:46  execl.c
-rw-rw-r-- 1 paleti paleti 1.1K Sep  7 11:46  fib_prime.c
-rw-rw-r-- 1 paleti paleti 223K Sep  5 00:08 'Fork Assignment-2.pdf'
-rwxrwxr-x 1 paleti paleti 8.4K Sep 17 00:55  leadtrail
-rw-rw-r-- 1 paleti paleti 1.7K Sep 17 00:54  lead_trail_sort.c
-rw-rw-r-- 1 paleti paleti  555 Sep  8 10:59  oddeven_series_gen.c
-rw-rw-r-- 1 paleti paleti  654 Sep  7 21:05  oddevensum.c
-rw-rw-r-- 1 paleti paleti    2 Sep  8 01:32  temp.txt
Parent process gets control
Parent has waited for child to complete
paleti@paletil:~/OS_LAB/fork_2$
```

execlp()

execl() does not use the PATH environment variable. So, the full path of the executable file is required to run it with execl(). execlp() uses the PATH environment variable. So, if an executable file or command is available in the PATH, then the command or the filename is enough to run it, the full path is not needed.

I only passed the command name **ls**, not the full path **/bin/ls**. As you can see, I got the same output as before.

Paleti Krishnasai
CED18I039

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()

{
   pid_t pid; //this is to use the pid data type
   pid = fork();
   if (pid == 0)
       execlp("ls", "ls", "-lh", NULL); //child image is now ls command
   else
   {
       wait(NULL); // parent waits for the child to complete execution
       printf("Parent process gets control \n");
       printf("Parent has waited for child to complete\n");
   }
}
```

Output:

```
paleti@paletil:~/OS_LAB/fork_2$ ./execlp
total 332K
-rwxrwxr-x 1 paleti paleti  13K Sep  8 21:47  a.out
-rw-rw-r-- 1 paleti paleti 1.3K Sep  8 01:32  armstrong.c
-rwxrwxr-x 1 paleti paleti 8.5K Sep 17 00:52  armstronggen
-rw-rw-r-- 1 paleti paleti 1.1K Sep 17 00:52  armstrong_gen.c
-rw-rw-r-- 1 paleti paleti 2.0K Sep  8 21:47  ascdescsort.c
-rw-rw-r-- 1 paleti paleti  197 Sep 18 00:44  bsearch.cpp
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 00:46  execl
-rw-rw-r-- 1 paleti paleti  487 Sep 18 00:46  execl.c
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 00:54  execlp
-rw-rw-r-- 1 paleti paleti  483 Sep 18 00:51  execlp.c
-rw-rw-r-- 1 paleti paleti 1.1K Sep  7 11:46  fib_prime.c
-rw-rw-r-- 1 paleti paleti 223K Sep  5 00:08 'Fork Assignment-2.pdf'
-rwxrwxr-x 1 paleti paleti 8.4K Sep 17 00:55  leadtrail
-rw-rw-r-- 1 paleti paleti 1.7K Sep 17 00:54  lead_trail_sort.c
-rw-rw-r-- 1 paleti paleti  555 Sep  8 10:59  oddeven_series_gen.c
-rw-rw-r-- 1 paleti paleti  654 Sep  7 21:05  oddevensum.c
-rw-rw-r-- 1 paleti paleti    2 Sep  8 01:32  temp.txt
Parent process gets control
Parent has waited for child to complete
paleti@paletil:~/OS_LAB/fork_2$
```

Paleti Krishnasai
CED18I039

execv()

In execl() function, the parameters of the executable file are passed to the function as different arguments. With execv(), you can pass all the parameters in a NULL terminated array argv. The first element of the array should be the path of the executable file. Otherwise, execv() function works just as execl() function.

As you can see, I am getting the correct output.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()

{
   pid_t pid; //this is to use the pid data type
   char *args[] = {"/bin/ls", "-lh", NULL};
   pid = fork();
   if (pid == 0)
       execv("/bin/ls",args); //child image is now ls command
   else
   {
       wait(NULL); // parent waits for the child to complete execution
       printf("Parent process gets control \n");
       printf("Parent has waited for child to complete\n");
   }
}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/fork_2$ make execv
cc     execv.c   -o execv
paleti@paletil:~/OS_LAB/fork_2$ ./execv
total 348K
-rwxrwxr-x 1 paleti paleti  13K Sep  8 21:47  a.out
-rw-rw-r-- 1 paleti paleti 1.3K Sep  8 01:32  armstrong.c
-rwxrwxr-x 1 paleti paleti 8.5K Sep 17 00:52  armstronggen
-rw-rw-r-- 1 paleti paleti 1.1K Sep 17 00:52  armstrong_gen.c
-rw-rw-r-- 1 paleti paleti 2.0K Sep  8 21:47  ascdescsort.c
-rw-rw-r-- 1 paleti paleti  197 Sep 18 00:44  bsearch.cpp
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 00:46  execl
-rw-rw-r-- 1 paleti paleti  487 Sep 18 00:46  execl.c
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 00:54  execlp
-rw-rw-r-- 1 paleti paleti  483 Sep 18 00:51  execlp.c
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 01:25  execv
-rw-rw-r-- 1 paleti paleti  518 Sep 18 01:25  execv.c
-rw-rw-r-- 1 paleti paleti 1.1K Sep  7 11:46  fib_prime.c
-rw-rw-r-- 1 paleti paleti 223K Sep  5 00:08 'Fork Assignment-2.pdf'
-rwxrwxr-x 1 paleti paleti 8.4K Sep 17 00:55  leadtrail
-rw-rw-r-- 1 paleti paleti 1.7K Sep 17 00:54  lead_trail_sort.c
-rw-rw-r-- 1 paleti paleti  555 Sep  8 10:59  oddeven_series_gen.c
-rw-rw-r-- 1 paleti paleti  654 Sep  7 21:05  oddevensum.c
-rw-rw-r-- 1 paleti paleti    2 Sep  8 01:32  temp.txt
Parent process gets control
Parent has waited for child to complete
paleti@paletil:~/OS_LAB/fork_2$
```

execvp()

Works the same way as execv() system function. But, the PATH environment variable is used. So, the full path of the executable file is not required just as in execlp().

As you can see, the correct output is displayed.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()

{
   pid_t pid; //this is to use the pid data type
   char *args[] = {"ls", "-lh", NULL};
   pid = fork();
```

Paleti Krishnasai
CED18I039

```
    if (pid == 0)
        execvp("ls", args); //child image is now ls command
    else
    {
        wait(NULL); // parent waits for the child to complete execution
        printf("Parent process gets control \n");
        printf("Parent has waited for child to complete\n");
    }
}
```

Output:

```
cc     execvp.c   -o execvp
paleti@paletil:~/OS_LAB/fork_2$ ./execvp
total 364K
-rwxrwxr-x 1 paleti paleti  13K Sep  8 21:47  a.out
-rw-rw-r-- 1 paleti paleti 1.3K Sep  8 01:32  armstrong.c
-rwxrwxr-x 1 paleti paleti 8.5K Sep 17 00:52  armstronggen
-rw-rw-r-- 1 paleti paleti 1.1K Sep 17 00:52  armstrong_gen.c
-rw-rw-r-- 1 paleti paleti 2.0K Sep  8 21:47  ascdescsort.c
-rw-rw-r-- 1 paleti paleti  197 Sep 18 00:44  bsearch.cpp
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 00:46  execl
-rw-rw-r-- 1 paleti paleti  487 Sep 18 00:46  execl.c
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 00:54  execlp
-rw-rw-r-- 1 paleti paleti  483 Sep 18 00:51  execlp.c
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 01:25  execv
-rw-rw-r-- 1 paleti paleti  518 Sep 18 01:25  execv.c
-rwxrwxr-x 1 paleti paleti 8.3K Sep 18 01:29  execvp
-rw-rw-r-- 1 paleti paleti  510 Sep 18 01:29  execvp.c
-rw-rw-r-- 1 paleti paleti 1.1K Sep  7 11:46  fib_prime.c
-rw-rw-r-- 1 paleti paleti 223K Sep  5 00:08 'Fork Assignment-2.pdf'
-rwxrwxr-x 1 paleti paleti 8.4K Sep 17 00:55  leadtrail
-rw-rw-r-- 1 paleti paleti 1.7K Sep 17 00:54  lead_trail_sort.c
-rw-rw-r-- 1 paleti paleti  555 Sep  8 10:59  oddeven_series_gen.c
-rw-rw-r-- 1 paleti paleti  654 Sep  7 21:05  oddevensum.c
-rw-rw-r-- 1 paleti paleti    2 Sep  8 01:32  temp.txt
Parent process gets control
Parent has waited for child to complete
```

Paleti Krishnasai
CED18I039

(2) Odd and Even series generation for n terms using Parent Child relationship (say odd is the duty of the parent and even series as that of child)

Approach:
This is a simple odd and even series setup with WAIT() being used in parent process for the child to finish the execution so as to avoid interleaving prints as the scheduling is based on the kernel.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
   int n;
   printf("Enter the size : \n");
   scanf("%d",&n);

   int pid;
   pid = fork();

   if(pid == 0)
   {
       printf("Even Series : \n");
       for(int i=0;i<n;i++)
       printf("%d\n",2*i);
       printf("\n");
   }
   else if(pid>0)
   {
       wait(NULL);

       printf("Odd series : \n");
       for(int i=0;i<n;i++)
       printf("%d\n",2*i+1);
```

```
        printf("\n");
    }


    return 0;
}
```

Output:

```
paleti@paletil:~/OS_LAB/fork_2$ ./a.out
Enter the size :
10
Even Series :
0
2
4
6
8
10
12
14
16
18

Odd series :
1
3
5
7
9
11
13
15
17
19
```

(b) given a series of n numbers (u can assume natural numbers till n) generate the sum of odd terms in the parent and the sum of even terms in the child process.

Approach:

This is an extension of the previous code where in we are adding the numbers before printing.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Paleti Krishnasai
CED18I039

```c
int main()
{
   int n;
   int sum =0;
   printf("Enter the size : \n");
   scanf("%d", &n);

   int pid;
   pid = fork();

   if (pid == 0)
   {

       printf("Even Series : \n");
       for (int i = 0; i < n; i++)
       sum = sum + (2*i) ;
       printf("%d\n",sum);
       printf("\n");
   }
   else if (pid > 0)
   {
       wait(NULL);

       printf("Odd series : \n");
       for (int i = 0; i < n; i++)
       sum = sum + (2*i+1);
       printf("%d\n",sum);


       printf("\n");
   }

   return 0;
}
```

Paleti Krishnasai
CED18I039

Output :

```
paleti@paletil:~/OS_LAB/fork_2$ gcc oddevensum.c
paleti@paletil:~/OS_LAB/fork_2$ ./a.out
Enter the size :
5
Even Series :
20

Odd series :
25
```

(3) Armstrong number generation within a range. The digit extraction, cubing can be the responsibility of the child while the checking for sum == no can happen in the child and the output list in the child.

Approach:

Only the Narcissistic numbers of order 3 are allowed in this program as the code is written for classic armstrong numbers. The cubed result of the digits is stored in an array as VFORK() has been used for memory sharing .

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Paleti Krishnasai
CED18I039

```c
int main()

{
   pid_t pid;
   int number,remainder;
   int start,end;
   printf("Enter the starting number\n");
   scanf("%d",&start);
   printf("Enter the ending number\n");
   scanf("%d",&end);
   int range = end-start+1;
   int result[range];
   for(int i=0;i<range;i++)
   result[i]=0;

   pid = vfork();

   if (pid == 0)
   {
       for (int i = start; i < end+1; i++)
       {
           number = i;
           do
           {
               remainder = number % 10;
               number = number / 10 ;
               result[i-start] += pow(remainder,3);
           }while (number != 0);
       }
       exit(0);
   }
   else if( pid >0 )
   {
       wait(NULL);
       for (int i = start; i <=end; i++)
       {
           if (result[i-start]==i)
```

```
            printf("%d is an armstrong number\n",i);
        }


    }

      return 0;
}
```

Output :

```
paleti@paletil:~/OS_LAB/fork_2$ gcc armstrong_gen.c -lm -o armstrongg
en
paleti@paletil:~/OS_LAB/fork_2$ ./armstronggen
Enter the starting number
0
Enter the ending number
1000
0 is an armstrong number
1 is an armstrong number
153 is an armstrong number
370 is an armstrong number
371 is an armstrong number
407 is an armstrong number
paleti@paletil:~/OS_LAB/fork_2$
```

(4) Fibonacci Series AND Prime parent child relationship (say parent does fib Number generation using series and child does prime series)

Approach:
Parent takes fib and child take prime series generation .The fibonacci series is generated using an unconventional approach of using 2 variables and updating them in a for loop for the desired numbers.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
```

Paleti Krishnasai
CED18I039

```c
#include <unistd.h>

int main()
{
   int size;
   printf("Enter the size: \n");
   scanf("%d",&size);
   int pid;
   pid = fork();


   if(pid ==0)
   {
       printf("Prime Series : \n");
       int n=2;
       int no=0;
       while(no < size)
       {
           int a = n-1;
           while(a>1)
           {
               if(n%a == 0)
               {
                   a=-1;
                   break;
               }
               a--;
           }
           if(a>0)
           {
               printf("%d\n",n);
               no++;
           }
           n++;
       }
       printf("\n");
```

Paleti Krishnasai
CED18I039

```
    }
    else if(pid > 0)
    {
        wait(NULL);
        printf("Fibonacci series : \n");
        int a= 1,b =1;
        printf("1\n1\n");
        for(int i=0;i<size-2;i++)
        {
            a = a + b;
            b = a - b;
            printf("%d\n",a);
        }
        printf("\n");

    }

    return 0;
}
```

Output :

```
paleti@paletil:~/OS_LAB/fork_2$ gcc fib_prime.c
paleti@paletil:~/OS_LAB/fork_2$ ./a.out
Enter the size:
5
Prime Series :
2
3
5
7
11

Fibonacci series :
1
1
2
3
5
```

Paleti Krishnasai
CED18I039

(5) Ascending Order sort within Parent and Descending order sort (or vice versa) within the child
process of an input array. (u can view as two different outputs –first entire array is asc order sorted in op and then the second part desc order output)

Approach:
Bubble sort has been used to sort the list in parent and child processes respectively.

Code:

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/types.h>

int main()
{


   int size = 0;
   printf("Enter size of array: ");
   scanf("%d", &size);

   printf("Enter the array: ");
   int arr[size];
   for (int i = 0; i < size; i++)
   {
       scanf("%d", &arr[i]);
   }

   int pid = fork();

   if (pid == 0)
   {
       printf("\nDescending order: ");
```

Paleti Krishnasai
CED18I039

```
    int pid2 = vfork();

    if (pid2 == 0)
    {
        for (int i = 0; i < size; i++)
        {
            int max_i = i;
            for (int j = i; j < size; j++)
            {
                if (arr[j] > arr[max_i])
                {
                    max_i = j;
                }
            }
            int temp = arr[max_i];
            arr[max_i] = arr[i];
            arr[i] = temp;
        }
        exit(0);
    }
    else if (pid2 > 0)
    {
        wait(NULL);
        for (int i = 0; i < size; i++)
            printf("%d ", arr[i]);

        printf("\n");
        return 0;
    }
}
else if (pid > 0)
{
    wait(NULL);
    printf("\nAscending order: ");

    int pid2 = vfork();
```

Paleti Krishnasai
CED18I039

```
        if (pid2 == 0)
        {
            for (int i = 0; i < size; i++)
            {
                int max_i = 0;
                for (int j = 0; j < size - i; j++)
                {
                    if (arr[j] > arr[max_i])
                    {
                        max_i = j;
                    }
                }

                int temp = arr[max_i];
                arr[max_i] = arr[size - i - 1];
                arr[size - i - 1] = temp;
            }
            exit(0);
        }
        else if (pid2 > 0)
        {
            wait(NULL);
            for (int i = 0; i < size; i++)
                printf("%d ", arr[i]);

            printf("\n");

            return 0;
        }
    }
}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/fork_2$ gcc ascdescsort.c
paleti@paletil:~/OS_LAB/fork_2$ ./a.out
Enter size of array: 6
Enter the array: 8 9 6 3 6 2

Descending order: 9 8 6 6 3 2

Ascending order: 2 3 6 6 8 9
```

(6) Given an input array use parent child relationship to sort the first half of array in ascending order and the trailing half in descending order (parent / child is ur choice)

Approach:
Bubble sort has been used to sort the list and a flag variable is used to cut the list into 2 parts.

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{

   int size = 0;
   printf("Enter size of array: ");
   scanf("%d", &size);

   int mid = size / 2 - 1;
```

Paleti Krishnasai
CED18I039

```
printf("Enter the array: ");
int arr[size];
for (int i = 0; i < size; i++)
{
    scanf("%d", &arr[i]);
}

int pid = fork();

if (pid == 0)
{
    printf("Sorted array: ");
    //printf("Leading Ascending order: ");

    for (int i = 0; i < mid + 1; i++)
    {
        int max_i = 0;
        for (int j = 0; j < mid + 1 - i; j++)
        {
            if (arr[j] > arr[max_i])
            {
                max_i = j;
            }
        }
        int temp = arr[max_i];
        arr[max_i] = arr[mid + 1 - i - 1];
        arr[mid + 1 - i - 1] = temp;
    }

    for (int i = 0; i < mid + 1; i++)
        printf("%d ", arr[i]);

    //printf("\n");
    printf(" | ");
    return 0;
}
else if (pid > 0)
```

Paleti Krishnasai
CED18I039

```
    {
        wait(NULL);
        //printf("Trailing Descending order: ");

        for (int i = mid + 1; i < size; i++)
        {
            int max_i = i;
            for (int j = i; j < size; j++)
            {
                if (arr[j] > arr[max_i])
                {
                    max_i = j;
                }
            }
            int temp = arr[max_i];
            arr[max_i] = arr[i];
            arr[i] = temp;
        }

        for (int i = mid + 1; i < size; i++)
            printf("%d ", arr[i]);

        printf("\n");
        return 0;
    }
}
```

Paleti Krishnasai
CED18I039

Output :

```
paleti@paletil:~/OS_LAB/fork_2$ gcc lead_trail_sort.c -o leadtrail
paleti@paletil:~/OS_LAB/fork_2$ ./leadtrail
Enter size of array: 6
Enter the array: 1 9 8 6 3
5
Sorted array: 1 8 9  | 6 5 3
paleti@paletil:~/OS_LAB/fork_2$ ./leadtrail
Enter size of array: 5
Enter the array: 1 5 4 6 3
Sorted array: 1 5  | 6 4 3
paleti@paletil:~/OS_LAB/fork_2$
```

(7) Implement a multiprocessing version of binary search where the parent searches for the key in the first half and subsequent splits while the child searches in the other half of the array. By default u can implement a search for the first occurrence and later extend to support multiple occurrence (duplicated elements search as well)

Approach:

Binary search with duplicate entry count is implemented where in the user gets the count of the key being searched , the count is shown according to the number of occurrences in the 2 parts of the list (first and second) . True parallelism of this method would have a lesser time to search though the asymptotic complexity remains the same. sort() function has been used to sort the parts of the input list.

Code:

```
#include <bits/stdc++.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
using namespace std;
```

Paleti Krishnasai
CED18I039

```
int main()
{
   int size;
   printf("Enter the size of the list : \n");
   scanf("%d",&size);
   int key;int count=0,count1=0;
   int arr[size];
   printf("Enter the list : \n");
   for(int i=0; i<size ;i++)
   scanf("%d",&arr[i]);
   printf("Enter the key : \n");
   scanf("%d",&key);
   int beg,end;
   int mid=size/2;

   pid_t pid;
   pid = vfork();

   if(pid==0)
   {
       sort(arr,arr+size/2);

       beg = 0;
       end = mid;
       for(int i=0;i<end;i++)
       printf("%d",arr[i]);
       printf("\n");
     //printf("%d  %d ",beg,end);
       while(beg<=end)
       {
           int mid_temp = (beg + end) / 2;
           //printf("%d\n",mid_temp);
           if(arr[mid_temp]==key)
           {
               count++;
               for(int i=mid_temp+1;i<end;i++)
               {
```

Paleti Krishnasai
CED18I039

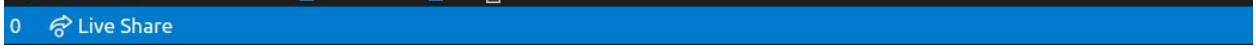
```
                if(arr[i]==key)
                count++;
                else break;
            }
            for(int i=beg;i<mid_temp;i++)
            {
                if(arr[i]==key)
                count++;
                else break;
            }
            break;
        }
        else if (arr[mid_temp] < key)
            beg = mid_temp + 1;
        else end = mid_temp-1;
    }
    printf("%d occurs %d times in first half\n",key,count);

    exit(0);
}

else if (pid > 0)
{
    wait(NULL);
    sort(arr+size/2,arr+size);
    beg=mid;
    end=size;
    for (int i = beg; i < end; i++)
        printf("%d", arr[i]);
    printf("\n");
    //printf("%d  %d ",beg,end);
    while (beg <= end)
    {
        int mid_temp = (beg + end) / 2;
        //printf("%d\n", mid_temp);
        if (arr[mid_temp] == key)
        {
```

```
                count1++;
                for (int i = mid_temp + 1; i < end; i++)
                {
                    if (arr[i] == key)
                        count1++;
                    else
                        break;
                }
                for (int i = beg; i < mid_temp; i++)
                {
                    if (arr[i] == key)
                        count1++;
                    else
                        break;
                }
                break;
            }
            else if (arr[mid_temp] < key)
                beg = mid_temp + 1;
            else
                end = mid_temp - 1;
        }
        printf("%d occurs %d times in second half\n", key, count1);
    }



    return 0;
}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/fork_2$ ./bsearch
Enter the size of the list :
6
Enter the list :
2 1 1 9 8 5
Enter the key :
5
112
5 occurs 0 times in first half
589
5 occurs 1 times in second half
paleti@paletil:~/OS_LAB/fork_2$ ./bsearch
Enter the size of the list :
6
Enter the list :
2 1 1 1 5 6
Enter the key :
1
112
1 occurs 2 times in first half
156
1 occurs 1 times in second half
paleti@paletil:~/OS_LAB/fork_2$ ./bsearch
Enter the size of the list :
5
Enter the list :
2 1 1 5 0
Enter the key :
1
12
1 occurs 1 times in first half
015
1 occurs 1 times in second half
paleti@paletil:~/OS_LAB/fork_2$
```



Paleti Krishnasai
CED18I039

(8) * Non Mandatory [extra credits]
Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a parent child relationship to contributes a faster version of fib series generation as opposed to sequential
logic in (4)

Approach 1(Iterative)

This approach is a generalization of the fibonacci series , but it has been implemented in a single process setup. As implementation in a 2 process setup has proven buggy and if the implementation works , CED18I039 will submit in a separate doc as version 2 of the code.

Code:

```
#include <bits/stdc++.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
using namespace std;

// A generalized equation for fibonacci
// F(n+k) = F(n+1)*F(k) + F(n)* F(k-1)
// Explanation:
/*
   F(n+2)=F(n+1)+F(n)          ->(1)
   F(n+3)=F(n+2)+F(n+1)
         =F(n+1)+F(n+1)+F(n) [from (1)]
         =F(n+1)*2 + F(n)*1    ->(2)
   F(n+4)=F(n+3)+F(n+2)
         =F(n+1)*2 + F(n)*1 +F(n+1) + F(n) [from (1 & 2)]
```

Paleti Krishnasai
CED18I039

```
         =F(n+1)*3 + F(n)*2
   F(0) = 0
   F(1) = 1
   F(2) = 1
   F(3) = 2
   F(4) = 3
   Hence F(n+k) = F(n+1)*F(k) + F(n)*F(k-1)
*/
int main()

{
   pid_t pid;
   int size ;
   printf("Enter the size : \n");
   scanf("%d",&size);
   int fib[size+1];
   fib[0] = 0;
   fib[1] = 1;
   fib[2] = 1;
   //now from the formula , to simulate in parent child we can take n=1
   printf("1 : %d\n2 : %d\n",fib[1],fib[2]);
   pid = vfork();

   if(pid == 0)
   {
       for(int i=2;i<size;++i)
       {
           fib[i + 1] = fib[2] * fib[i] + fib[1] * fib[i - 1];
           printf("%d : %d\n",i+1,fib[i+1]);
       }
       exit(0);

   }
   else if(pid > 0)
   {
       wait(NULL);
       printf("Prime Series : \n");
```

Paleti Krishnasai
CED18I039

```
        int n = 2;
        int no = 0;
        while (no < size)
        {
            int a = n - 1;
            while (a > 1)
            {
                if (n % a == 0)
                {
                    a = -1;
                    break;
                }
                a--;
            }
            if (a > 0)
            {
                printf("%d\n", n);
                no++;
            }
            n++;
        }
        printf("\n");
    }



    return 0;
}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/fork_2$ make FibPrll
g++     FibPrll.cpp   -o FibPrll
paleti@paletil:~/OS_LAB/fork_2$ ./FibPrll
Enter the size :
6
1 : 1
2 : 1
3 : 2
4 : 3
5 : 5
6 : 8
Prime Series :
2
3
5
7
11
13
```

Approach 2(Recursive)

In this approach the user has tried fibonacci in a 2 process setup with recursively calling the function starting from n=1 as for fibonacci to work we need the previous entry values .

Code:

```
#include <bits/stdc++.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
```

Paleti Krishnasai
CED18I039

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
using namespace std;
int fib(int n)
{
   if(n==0 || n==1)
   return n;
   else
   {
       int sum =0;
       pid_t pid;
       pid = vfork();

       if(pid == 0)
       {
           sum +=fib(n-1);
           exit(0);
       }
       else if(pid > 0)
       {
           wait(NULL);
           sum+=fib(n-2);
           return sum;
       }

   }

}
int main()

{
   int size;
   printf("Enter the size : \n");
   scanf("%d",&size);
   printf("\n");
   for(int i=1;i<=size;i++)
```

```
    printf("%d\n",fib(i));
    return 0;
}
```

Output:

```
paleti@paletil:~/OS_LAB/fork_2$ ./fibp
Enter the size :
6

1
1
2
3
5
8
paleti@paletil:~/OS_LAB/fork_2$ ./fibp
Enter the size :
10

1
1
2
3
5
8
13
21
34
55
```